periodica polytechnica

Electrical Engineering
55/1-2 (2011) 31–43
doi: 10.3311/pp.ee.2011-1-2.04
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

Formal verification of secure ad-hoc network routing protocols using deductive model-checking

Levente Buttyán / *Ta Vinh* Thong

Received 2011-06-06

Abstract

Ad-hoc networks do not rely on a pre-installed infrastructure, but they are formed by end-user devices in a self-organized manner. A consequence of this principle is that end-user devices must also perform routing functions. However, end-user devices can easily be compromised, and they may not follow the routing protocol faithfully. Such compromised and misbehaving nodes can disrupt routing, and hence, disable the operation of the network. In order to cope with this problem, several secured routing protocols have been proposed for ad-hoc networks. However, many of them have design flaws that still make them vulnerable to attacks mounted by compromised nodes. In this paper, we propose a formal verification method for secure ad-hoc network routing protocols that helps increasing the confidence in a protocol by providing an analysis framework that is more systematic, and hence, less error-prone than the informal analysis. Our approach is based on a new process calculus that we specifically developed for secure ad-hoc network routing protocols and a deductive proof technique. The novelty of this approach is that contrary to prior attempts to formal verification of secure ad-hoc network routing protocols, our verification method can be made fully automated.

Keywords

Formal verification · routing protocol · security · ad-hoc activals · process calculus · automatic reasoning.

Acknowledgement

The work described in this paper has been supported by the High Speed Networks Laboratory (HSN Lab) at the Budapest University of Technology and Economics. The first author has been further supported by the Hungarian Academy of Sciences through the Bolyai János Research Fellowship.

Levente Buttyán

Department of Telecommunications, BME, H-1117 Budapest, Magyar tud. krt. 2., Hungary

Ta Vinh Thong

Department of Electronics Technology, BME, H-1117 Budapest, Magyar tud. krt. 2., Hungary

1 Introduction

In the recent past, the idea of ad-hoc networks have created a lot of interest in the research community, and it is now starting to materialize in practice in various forms, ranging from static sensor networks through opportunistic interactions between personal communication devices to vehicular networks with increased mobility. A common property of these systems is that they have sporadic access, if at all, to fixed, pre-installed communication infrastructures. Hence, it is usually assumed that the devices in ad-hoc networks play multiple roles: they are terminals and network nodes at the same time.

In their role as network nodes, the devices in ad-hoc networks perform basic networking functions, most notably routing. At the same time, in their role as terminals, they are in the hands of end-users, or they are installed in physically easily accessible places. In any case, they can be easily compromised and re-programmed such that they do not follow the routing protocol faithfully. The motivations for such re-programming could range from malicious objectives (e.g., to disrupt the operation of the network) to selfishness (e.g., to save precious resources such as battery power). The problem is that such compromised and misbehaving routers may have a profound negative effect on the performance of the network.

In order to mitigate the effect of misbehaving routers on network performance, a number of secured routing protocols have been proposed for ad-hoc networks (see e.g., [6] for a survey). These protocols use various mechanisms, such as cryptographic coding, multi-path routing, and anomaly detection techniques, to increase the resistance of the protocol against attacks. Unfortunately, the design of secure routing protocols is an error-prone activity, and indeed, most of the proposed secure ad-hoc network routing protocols turned out to be still vulnerable to attacks. This fact implies that the design of secure ad-hoc network routing protocols should be based on a systematic approach that minimizes the number of mistakes made in the design.

As an important step towards this goal, in this paper, we propose a formal method to verify the correctness of secure ad-hoc network routing protocols. Our approach is based on a new process calculus that we specifically developed for modeling the op-

eration of secure ad-hoc network routing protocols, and a proof technique based on deductive model checking. The systematic nature of our method and its well-founded semantics ensure that one can have much more confidence in a security proof obtained with our method than in a "proof" based on informal arguments. In addition, compared to previous approaches that attempted to formalize the verification process of secure ad-hoc network routing protocols [3, 11–13], the novelty of our approach is that it can be fully automated.

The organization of the paper is the following: In Section 2, we describe the SRP protocol [14] as an example for a secure ad-hoc network routing protocol that we will use for illustration purposes throughout the paper. In Section 3 we give an overview of the most relevant related works. In Section 4, we introduce the syntax and the semantics of our process calculus. In Section 5, we demonstrate the expressive power of our calculus by modeling the operation of SRP and by formally representing a known security flaw in SRP. In Section 6, we discuss how the verification process can be automated and describe our deductive proof technique. In addition, we demonstrate the application of our method via an example. Finally, in Section 7, we conclude the paper and discuss our planned future work on this topic.

2 The SRP protocol and an attack on it

SRP is a secure on-demand source routing protocol for ad-hoc networks proposed in [14]. The design of the protocol is inspired by the DSR protocol [15], however, DSR has no security mechanisms at all. Thus, SRP can be viewed as a secure variant of DSR. SRP tries to cope with attacks by using a cryptographic checksum in the routing control messages (route requests and route replies). This checksum is computed with the help of a key shared by the initiator and the target of the route discovery process; hence, SRP assumes only shared keys between communicating pairs.

In SRP, the initiator of the route discovery generates a route request message and broadcasts it to its neighbors. The integrity of this route request is protected by a Message Authentication Code (MAC) that is computed with a key shared by the initiator and the target of the discovery. Each intermediate node that receives the route request for the first time appends its identifier to the request and re-broadcasts it. The MAC in the request is not updated by the intermediate nodes, as by assumption, they do not necessarily share a key with the target. When the route request reaches the target of the route discovery, it contains the list of identifiers of the intermediate nodes that passed the request on. This list is considered as a route found between the initiator and the target.

The target verifies the MAC of the initiator in the request. If the verification is successful, then it generates a route reply and sends it back to the initiator via the reverse of the route obtained from the route request. The route reply contains the route obtained from the route request, and its integrity is protected by another MAC generated by the target with a key shared by the target and the initiator. Each intermediate node passes the route reply to the next node on the route (towards the initiator) without modifying it. When the initiator receives the reply it verifies the MAC of the target, and if this verification is successful, then it accepts the route returned in the reply.

The basic problem in SRP is that the intermediate nodes cannot check the MAC in the routing control messages. Hence, compromised intermediate nodes can manipulate control messages, such that the other intermediate nodes do not detect such manipulations. Furthermore, the accumulated node list in the route request is not protected by the MAC in the request, hence it can be manipulated without the target detecting such manipulations.

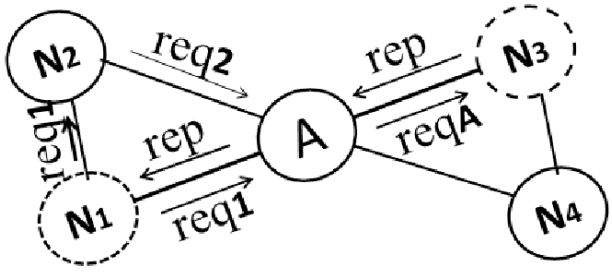

Fig. 1. An attack scenario against the SRP protocol.

In order to illustrate a known attack on SRP, let us consider the network topology shown in Fig. 1. Let us further assume that node N_1 initiates a route discovery to node N_3. The attacker node A can manipulate the accumulated list of node identifiers in the route request such that N_3 receives the request with the list (N_2, n, N_4), where n is an arbitrary fake identifier. This manipulation remains undetected, because the MAC computed by N_1 does not protect the accumulated node list in the route request, and intermediate nodes do not authenticate the request. When the target N_3 sends the route reply, A forwards it without modification to N_1 in the name of N_2. As the route reply is not modified, the MAC of the target N_3 verifies correctly at N_1, and thus, N_1 accepts the route (N_1, N_2, n, N_4, N_3). However, this is a mistake, because there is no link between N_2 and n, and between n and N_4.

Note that the above attack has been found by manual analysis of the protocol. However, there may be many similar attack scenarios (and indeed, there are), and manual analysis would be inefficient to find all of them. The very purpose of our formal verification method to be introduced in the upcoming sections of the paper is to make the analysis systematic and amenable for automation such that it can efficiently find all possible attacks against a protocol (within some limits of the underlying model).

3 Related works

Our purpose in this paper is to provide a formal modelling method for secure on-demand source routing protocols and a systematic and automatic method for detecting attacks similar to the attack we introduced in the previous section. Till now there

are only few works addressing this problem. Each method proposed in the most important related works [1–3, 7, 8, 12, 17–19] has numerous drawbacks that we will discuss in the following:

In works [3, 12] the authors model the operation of the protocol participants by interactive and probabilistic Turing machines, where the interaction is realized via common tapes. This model enables us to be concerned with arbitrary feasible attacks. The security objective function is applied to the output of this model (i.e., the final state of the system) in order to decide whether the protocol functions correctly or not. Once the model is defined, the goal is to prove that for any adversary, the probability that the security objective function is not satisfied is negligible. The main drawback of this method is that the proof is not systematic and automated, and the framework is not well-suited for detecting attack scenarios once the proof fails. In this paper we aim at improving these works by adding automated verification method based on deductive model-checking.

In [10] the authors present the *applied π-calculus* that is a variant of the pure *π-calculus* [4]. The applied π-calculus is well-suited for modelling security protocols because it provides expressive syntax and semantics for reasoning of cryptographyic primitives and operations. However, it lacks syntax and semantics for reasoning of broadcast communication, neighborhood, and communication range. Therefore, the applied π-calculus cannot be used direcly for modelling routing protocols.

In order to give a formal and precise mathematical reasoning of the operation of routing protocols several process calculi have been proposed in the recent years. Among them the two works [1, 2] are closest to our work.

In the works [1, 17] the author proposes the process calculus, CMAN, for modelling mobile ad-hoc networks. The advantage of CMAN is that it includes syntax and semantics for modelling cryptographic primitives, neighborhood, broadcast communication. The main drawback of CMAN is that it does not provide syntax and semantics for modelling the accumulated knowledge of the attacker node, therefore, it cannot be directly used to model the attack scenario against SRP protocol we showed in the Section 2 or the similar attacks presented in [3]. In these attacks the attacker node waits and collects information it intercepts during the route discovery, which it uses later to construct messages that contain incorrect route. In order to model these kind of attacks we propose the notion of the *active substitution with range* in our calculus.

In [2] the authors propose the *ω-calculus*. The main advantage of this calculus is that it has syntax and semantics for neighborhood, broadcast communication and mobility. The main drawback of this method is it does not provide syntax and semantics for modelling cryptographic primitives and the attacker's knowledge base. In contrast to the ω-calculus our calculus cryptographic primitives and attacker's accumulated knowledge can be explicitly modelled.

The advantage of these process calculi is that the operation of mobile ad-hoc networks and several properties such as loop-freedom and security properties can be precisely and systematically described with them, however, the drawback of them is that the proofs and reasoning are still performed manually by hand.

Several works in the literature address the problem of automatic verification of routing protocols. In the works [18, 19] the authors investigate the problem of verifying loop freedom property of routing protocols. In [18] the LUNAR protocol is verified using the SPIN, and UPPAAL model-checkers; in [18] the authors verified the DYMO protocol using graph transformation. In contrast to these works we propose an automatic verification method focused on verifying *security properties* of "secure" routing protocols instead of loop freedom property.

The two works that are most related to our work are [7, 8]. In the works [7] and [8] the authors address the problem of verification security properties of secure routing protocols using the SPIN model-checker and CPAL-ES tools, respectively. The main drawbacks of these methods are that they suffer from expressiveness limitation. In particular, they cannot directly model cryptographic primitives and boadcast communications, instead they simulate cryptographic primites with a series of bites [7] and broadcast communication with a sequence of unicast communication. In contrast to these works, our automatic verification method provides a direct modelling of cryptography primitives and neighborhood.

4 Our calculus

In this section we define our calculus: Its syntax and informal semantics, as well as its operational semantics.

The advantage of our calculus is that its expressiveness allows for modelling broadcast communication, neighborhood, and transmission range like CMAN and the ω-calculus, and cryptographic primitives like the applied π-calculus, however, compared to them it includes novelties such as the definition of *active substitution with range* that enables us to model the attacker's knowledge base and attacks in the context of wireless ad-hoc networks. More precisely, our calculus can be seen as the combination of the three calculi with some extensions. More details about the novelties can be found in the technical report [16].

4.1 Syntax

We assume an infinite set of *names* $\mathcal{N}$ and *variables* $\mathcal{V}$, where $\mathcal{N} \cap \mathcal{V} = \emptyset$. Further, we define a set of *node identifiers* (node ID) $\mathcal{L}$, where $\mathcal{N} \cap \mathcal{L} = \emptyset$. Each node identifier uniquely identifies a node. Below the definition of *term*, denoted by t, is given:

$$t ::= req \mid rep \mid RouteOK \mid undef \mid true \mid c \mid n \mid l \mid x \mid f(t_1, \ldots, t_k).$$

Terms take their values from a set of data of different types, namely

- *req* and *rep* are unique constants that represent the *req* and *rep* tags in route request and reply messages;

- *RouteOK* is a special constant. The source node outputs *RouteOK* when it receives the reply message and all the verifications it makes on it are successful.
- *undef* and *true* are special constants that we use during the analysis of the SRP protocol. More details will be given in Section 5.
- c models communication channels for unicast communication; n models some data;
- l models a node ID; x is a variable that models any term.
- Finally, f is a function with arity k defined on terms. Hence, the definition can be seen as a recursive definition. Function is used to model cryptographic primitives, route request and reply messages. For instance, digital signature is modelled by the function $sign(t_1, t_2)$, where t_1 models the message to be signed and t_2 models the secret key. Route request and reply messages are modelled by the function *tuple* of k terms $tuple(t_1,\ldots, t_k)$, which we abbreviate as $(t_1,\ldots, t_k)$.

The internal operation of nodes is modelled by *processes*. Processes can be inductively defined with the following syntax:

- The process $\overline{c}\langle t\rangle.P$ represents the unicast sending of message t on channel c, followed by the execution of P.
- The process $c(x).P$ receives some message on channel c and binds it to every variable x occurs in process P.
- The process $\langle t\rangle.P$ represents the broadcast send of message t, continued with the process P.
- The process $(x).P$ represents the broadcast receive of some message which will be bound to each occurence of x in P. Compared to the unicast case the two broadcast processes do not contain any certain channel, which intends to model that there is not a specified target.
- Processes $[t_1 = t_2]P$ and $[l \in \sigma]P$ mean that if $t_1 = t_2$ and $l \in \sigma$, respectively, then process P is "activated", else it stucks and idle.
- The *nil* process $\mathbf{0}$ does nothing.
- Process $let\ t = x\ in\ P$ represents the binding of term t to every variable x that occurs in process P.
- Process $P|Q$ is the *parallel composition* of processes P and Q and behaves as processes P and Q running in parallel: each may interact with the other, or with the outside world, independently of the other.
- Finally, process $!P$ represents the *infinite replication* of process P.

Nodes are defined as $\lfloor P\rfloor_l^\sigma$ which represents a node that has the identifier l and behaves as P, and its transmission range covers the nodes whose identifiers are in the set σ. Two nodes are neighbors if they are in each other's range. We note that σ can be empty and a node is $\lfloor P\rfloor_l$, which means that the node has no connections.

A *network*, denoted as N, can be an empty network with no nodes: 0_N; a singleton network with one node: $\lfloor P\rfloor_l$; the parallel composition of nodes: $\lfloor P_1\rfloor_{l_1}^{\sigma_1} \mid \lfloor P_2\rfloor_{l_2}^{\sigma_2}$, where σ_1 and σ_2 may include l_2 and l_1 respectively; and the composition of networks: $N_1 \mid N_2$.

Finally, in order to model attackers that improve their knowledge by accumulating information they hear from their neighbors, we extend the syntax of networks with the *active substitution with range*: An *extended network E* is defined as follows:

- An extended network can be a plain network N that we already discussed above.
- An extended network can be a parallel composition of two extended networks: $E_i|E_j$.
- An extended network is equipped with the active substitution with range $\{t/x\}^\sigma$. This says that the substitution $\{t/x\}$ binds term t to every variable x that occurs in any node $\lfloor P_i\rfloor_{l_i}^{\sigma_i}$ that is in parallel composition with $\{t/x\}^\sigma$, and $l_i \in \sigma$. Intuitively, σ is the range of the substitution $\{t/x\}$. Again, we note that the notion of active substitution with range is novel compared with CMAN, the ω-calculus and the applied π-calculus.

4.2 Semantics

The operational semantics of our calculus is defined by the *internal reduction* rules, *structural equivalences* and the labelled transition system (LTS) that composed of *labelled transition* rules. Due to space limitation in this paper we only introduce the most important rules. More details can be found in [16].

The internal reduction rules model the internal computation of nodes such as comparing two terms, while the labelled transition rules model the communication of nodes such as broadcast send and receive.

(Internal reduction rules)

(Red. Let) $\lfloor let\ x = t\ in\ P\rfloor_l^\sigma \rightarrow \lfloor P\{t/x\}\rfloor_l^\sigma$

(Red. If1) $\lfloor [t = t]P\rfloor_l^\sigma \rightarrow \lfloor P\rfloor_l^\sigma$

(Red. If2) $\lfloor [t = s]P\rfloor_l^\sigma \rightarrow \lfloor 0\rfloor_l^\sigma\ (if\ t \neq s)$

(Red. In1) $\lfloor [l \in \sigma]P\rfloor_l^\sigma \rightarrow \lfloor P\rfloor_l^\sigma\ (if\ l \in \sigma)$

(Red. In2) $\lfloor [l \in \sigma]P\rfloor_l^\sigma \rightarrow \lfloor 0\rfloor_l^\sigma\ (if\ l$ is not in $\sigma)$

The internal reduction relation is denoted by $\rightarrow$. Rule (Red. Let) models the binding of term t to variable x in process P. The rules (Red. If1) and (Red. If2) model the equality check of two terms. If the two terms are equal then the internal operation is followed by P, else it stucks; Finally, the rules (Red. In1) and (Red. In2) check if the node ID l is in the set σ.

In addition, internal reduction rules can be used to model mobility of nodes by the following rules:

(Reduction rules for mobility)

(Red. Connect)

$$\lfloor P \rfloor_{l_1}^{\sigma_1} | \lfloor Q \rfloor_{l_2}^{\sigma_2} \rightarrow_{\{l_1 \bullet\ l_2\}} \lfloor P \rfloor_{l_1}^{\sigma_1 \cup\ l_2} | \lfloor Q \rfloor_{l_2}^{\sigma_2 \cup\ l_1},$$

where l_2 is not in σ_1.

(Red. Disconnect)

$$\lfloor P \rfloor_{l_1}^{\sigma_1 \cup\ l_2} | \lfloor Q \rfloor_{l_2}^{\sigma_2 \cup\ l_1} \rightarrow_{\{l_1 \circ\ l_2\}} \lfloor P \rfloor_{l_1}^{\sigma_1} | \lfloor Q \rfloor_{l_2}^{\sigma_2},$$

where l_2 is not in σ_1.

The rule (Red. Connect) model the scenario in which node l_2 gets into the transmission range of the node l_1. This reduction relation is denoted by $\rightarrow_{\{l_1 \bullet\ l_2\}}$. Its counterpart is the rule (Red. Disconnect) says that node l_2 gets out of the transmission range of the node l_1. This reduction relation is denoted by $\rightarrow_{\{l_1 \circ\ l_2\}}$.

In the following, the most important labelled transitions are presented. Labelled transitions are used to model the communication of nodes, and play important roles in proofs. Labelled transitions are denoted by a labelled relation that is an arrow with a label: $\xrightarrow{\alpha}$.

(Labelled transition rules)

(Br-snd)

$$\lfloor \langle t \rangle.P \rfloor_l^{\sigma} \xrightarrow{\nu x.\langle x \rangle : \bar{l}\sigma} \{t/x\}^{\sigma} | \lfloor P \rfloor_l^{\sigma}$$

(Br-rcv)

$$\{t/x\}^{\sigma} | \lfloor (x).Q \rfloor_l^{\sigma_2} \xrightarrow{(t) : \{l \in \sigma\}} \{t/x\}^{\sigma} | \lfloor Q\{t/x\} \rfloor_l^{\sigma_2}$$

The first rule means that node l broadcasts t, so that t is now available for the nodes in its transmission range σ. This is modelled by the active substitution with range $\{t/x\}^{\sigma}$ and νx, which restricts the substitution to the nodes that are within the range σ. The second rule means that if the listening node l is within the range σ then it receives the broadcasted t.

Finally, next we introduce the most important structural equivalence rules on extended networks. The structural equivalence relation is denoted by $\equiv$. Using structural equivalence rules we can tell if two networks are identical up to structure.

(Struct. Equiv. rules for extended networks)

(Struct. E-Par1) $E | \mathbf{0}_N \equiv E$

(Struct. E-Par2) $E_1 | E_2 \equiv E_2 | E_1$

(Struct. E-Par3) $(E_1 | E_2) | E_3 \equiv E_1 | (E_2 | E_3)$

(Struct. E-Try) $\{t/x\}^{\sigma} | E \equiv \{t/x\}^{\sigma} | E\{t/x\}^{\sigma}$

(Struct. E-Rewrite) $\{t_1/x\}^{\sigma} \equiv \{t_2/x\}^{\sigma}$ (if $t_1 = t_2$)

The first rule says that the composition with an empty network does not change anything. The next two rules concern the commutative and associative properties. Rule (Struct. E-Rewrite) says that two active subtitutions with the same range σ and terms are stucturally equivalent. Rule (Struct. E-Try) tries to apply the active substitution with range $\{t/x\}^{\sigma}$ to the extended network E: For example, let E be

$$E = \{t_1/x_1\}^{\sigma_1} | \ldots | \{t_k/x_k\}^{\sigma_k} | \lfloor Q_i \rfloor_{l_i}^{\sigma_i} | \ldots | \left\lfloor Q_j \right\rfloor_{l_j}^{\sigma_j}$$

Then $E\{t/x\}^{\sigma}$ is

$$\{t_1/x_1\}^{\sigma_1} | \ldots | \{t_k/x_k\}^{\sigma_k} | \lfloor Q_i \rfloor_{l_i}^{\sigma_i} \{t/x\}^{\sigma} | \ldots | \lfloor Q_i \rfloor_{l_i}^{\sigma_i} \{t/x\}^{\sigma}$$

Intuitively, this means that the substitution is tried on every plain network. However, this substitution takes place at $\lfloor Q_i \rfloor_{l_i}^{\sigma_i}$ only if $l_i \in \sigma$. This is formally defined by a new rule (E-Subst) and the relation $\equiv_{l_i \in \sigma}$ below:

(Struct. E-Subst)

$$\{t/x\}^{\sigma} | \lfloor Q_i \rfloor_{l_i}^{\sigma_i} \equiv_{l_i \in \sigma} \lfloor Q_i\{t/x\} \rfloor_{l_i}^{\sigma_i}$$

We note that these labelled transition rules and structural equivalence rules are novel compared to the related works we mentioned above due to the application of the active substitution with range in each rule. In addition, the rules (Red. In1) and (Red. In2) are also new.

4.3 Labelled bisimilarity in context of wireless ad-hoc networks and attacker's knowledge base

In this subsection we give a definition of labelled bisimilarity that says if two wireless ad-hoc networks are equivalent, meaning that they cannot be distinguished by an observer which can eavesdrop on communications.

Let the extended network E be $\{t_1/x_1\}^{\sigma_1} | \ldots | \{t_n/x_n\}^{\sigma_n} | N_1 | \ldots | N_n$. The *frame* φ of E is the parallel composition $\{t_1/x_1\} | \ldots | \{t_n/x_n\}$ that models all the information that is output so far by the network E, which is $t_1, \ldots, t_n$ this case.

We say that two extended networks E_1 and E_2 are *statically equivalent*, denoted as $E_1 \approx_s E_2$, if their frames are statically equivalent. Two frames φ_1 and φ_2 are statically equivalent if they include the same number of active substitutions and same domain; and any two terms that are equal in φ_1 are equal in φ_2 as well. Intuitively, this means that the outputs of the two networks cannot be distinguished.

Definition 1 *Labelled bisimilarity ($\approx_l^N$) is the largest symmetric relation $\mathcal{R}$ on closed extended networks, such that $E_1 \mathcal{R} E_2$ implies: $\mathcal{L}(E_1) = \mathcal{L}(E_2)$, $C(E_1) = C(E_2)$, if*

- $E_1 \approx_s E_2$;
- *if $E_1 \rightarrow E_1'$, then $E_2 \rightarrow^* E_2'$ and $E_1' \mathcal{R} E_2'$ for some E_2';*
- *if $E_1 \xrightarrow{\nu x.\langle x \rangle : \bar{l}\sigma} E_1'$, then $\exists E_2'$ such that $E_2 \rightarrow^* \xrightarrow{\nu x.\langle x \rangle : \bar{l}\sigma} \rightarrow^* E_2'$ and $E_1' \mathcal{R} E_2'$ for some E_2'.*

Intuitively, this means that the outputs of the two networks of same topology cannot be distinguished during their operation. $\mathcal{L}(E)$ and $C(E)$ are the set of node IDs, and the neighborhood in E, respectively. In particular, the first condition means that at first E_1 and E_2 are statically equivalent; the second condition says that E_1 and E_2 remains statically equivalent after internal reduction steps. Finally, the third condition says that if the node l in E_1 outputs something then the node l in E_2 outputs the same thing, and the "states" E_1' and E_2' they reach after that remain statically equivalent. Here, $\rightarrow^*$ models the sequential execution of some internal reduction steps.

The **accumulated knowledge of the attacker** is defined as the frame with the identifier of the attacker l_a as parameter: $\varphi(l_a)$. The frame $\varphi(l_a)$ can be seen as the subset of the frame φ because it includes only such active substitutions $\{t_i/x_i\}^{\sigma_i}$ ($i \in \{1, \ldots, n\}$) where $l_a \in \sigma_i$. Formally,

$$\varphi(l_a) = \{t_i\ /_{x_i}\}^{\sigma_i} | \left\{t_j\ /_{x_j}\right\}^{\sigma_j} | \ldots | \{t_k\ /_{x_k}\}^{\sigma_k},$$

where $l_a \in \sigma_i, l_a \in \sigma_j, \ldots, l_a \in \sigma_k$, and $\{i, j, \ldots, k\} \subseteq \{1, 2 \ldots, n\}$.

5 Application of our calculus

Next we demonstrate the usabiliy of our calculus by modelling the SRP protocol and the attack shown in Section 2. The scenario in Section 2 is modelled by the extended network defined as:

$$netw \stackrel{def}{=} \lfloor P_1 \rfloor_{l_1}^{\{l_2,l_a\}} | \lfloor P_2 \rfloor_{l_2}^{\{l_1,l_a\}} | \lfloor !P_A \rfloor_{l_a}^{\{l_1,l_2,l_3,l_4\}} | \lfloor !P_3 \rfloor_{l_3}^{\{l_a,l_4\}} | \lfloor P_4 \rfloor_{l_4}^{\{l_a,l_3\}}$$

where the description of the nodes in the parallel compositions above corresponds to N_1, N_2, A, N_3, and N_4, respectively.

We use the following functions: The function $mac(t_1, t_2)$ computes the message authentication code of the message t_1 using the secret key t_2. The shared key between the nodes l_i and l_j is modelled by the function $k(l_i, l_j)$. The function $list(l_1, \ldots, l_n)$ models the list of node IDs, and $list()$ models the empty ID list. Functions $prev(List, l_i)$ and $next(List, l_i)$ return the element right before and after l_i in the list $List$, respectively; they return $undef$ if there is no any element before or after l_i. Function $toend(List, l_i)$ that appends l_i to the end of $List$. Functions $fst(List)$ and $lst(List)$ represent the first, and last element of $List$, respectively. Function $i((t_1, \ldots, t_n))$ returns the i-th ($i \in \{1,\ldots n\}$) element t_i of the tuple $(t_1, \ldots, t_n)$.

We note that in case of ad-hoc network, generally the behaviour of nodes can be specified in the same way meaning that every node can be a source, an intermediate node, and a destination node. However, for the sake of brevity, below we specifically consider the scenario in Section 2 and define the behaviour of nodes according to the scenario. This is sufficient to prove the vulnerability of SRP. The operation of the source node is specified as follows[1]:

$$\begin{aligned}
P_1 &\stackrel{def}{=} let\ MAC_{13} = mac\,((l_1, l_3), k(l_1, l_3))\ \ in\ Init.\\
Init &\stackrel{def}{=} \langle (req, l_1, l_3, MAC_{13}, list())\rangle.!Rep_1.\\
Rep_1 &\stackrel{def}{=} \left(x_{rep}\right).[1(x_{rep}) = l_1]\ [2(x_{rep}) = rep]\\
&\quad [3(x_{rep}) = l_1]\ [4(x_{rep}) = l_3]\\
&\quad [fst(5(x_{rep})) \in \{l_2, l_a\}]\\
&\quad [mac\left(\left(l_1, l_3, 5(x_{rep})\right), k(l_1, l_3)\right) = 6(x_{rep})]\\
&\quad \langle RouteOK\rangle.0
\end{aligned}$$

Intuitively, the first row models that the node l_1 computes the MAC using the key it shares with the destination node l_3. The second row means that node l_1 generates the route request message that includes the ID of the source and the target nodes, and the message authentication code MAC_{13}, then l_1 broadcasts it and waits for the reply. The exclamation mark models the infinite replication of Rep_1. Finally, the third row means that when l_1 receives a message, it checks whether (i) it is the addressee, (ii) the message is a reply, (iii) the ID of the source and the target nodes, and (iv) the message authentication code are correct. If all are correct then it signals the special constant $RouteOK$.

The description of the process P_2 is the following:

$$\begin{aligned}
P_2 &\stackrel{def}{=} \left(y_{req}\right).[1(y_{req}) = req].\\
&\langle\left(1(y_{req}), 2(y_{req}), 3(y_{req}), 4(y_{req}), toend(5(y_{req}), l_2)\right)\rangle.!Rep_2.\\
Rep_2 &\stackrel{def}{=} \left(y_{rep}\right).[1(y_{rep}) = l_2][2(y_{rep}) = rep]\\
&\quad [next\left(5(y_{rep}), l_2\right) \in \{l_1, l_a\}]\\
&\quad \langle\left(l_1, 2(y_{rep}), 3(y_{rep}), 4(y_{rep}), 5(y_{rep}), 6(y_{rep})\right)\rangle.0
\end{aligned}$$

Intuitively, on receiving a message node l_2 checks if it is a request, if so then node l_2 appends its ID to the end of the ID list, then re-broadcasts the request and waits for a reply. When it receives the reply message it checks (i) if the message is intended to it, (ii) the message is a reply, (iii) the ID next to l_2 in the list corresponds to the neighbors of node l_2 and forwards the reply to the source node l_1 if all verification steps pass.

Finally, the operation of the destination node is modelled as:

$$\begin{aligned}
P_3 &\stackrel{def}{=} \left(z_{req}\right).[1(z_{req}) = req][3(z_{req}) = l_3]\\
&\quad [mac\left(\langle 2(z_{req}), 3(z_{req})\rangle, k(l_1, l_3)\right) = 4(z_{req})].\\
&\quad let\ MAC_{31} = mac(\left(1(z_{req}), 2(z_{req}), 3(z_{req}), 5(z_{req})\right), k(l_1, l_3))in\\
&\quad let\ l_{prev} = lst\left(5(z_{req})\right)\ in\langle\left(l_{prev}, rep, 2(z_{req}), 3(z_{req}), 5(z_{req}), MAC_{31}\right)\rangle.0
\end{aligned}$$

Intuitively, on receiving a message node l_3 checks (i) if the message is a request, and (ii) it is the destination, and verifies the MAC embedded in the request using its shared key with l_1. If so then node l_3 creates a reply message and sends it back to the last node in the list.

The description of the node l_4 is the same as the node l_2 with the difference that it appends l_4 to the list instead of l_2. We refer the reader to [16] for details.

We give the model ($\mathcal{M}_A$) of the attacker node as follows: We assume that the attacker cannot forge the message authentication codes MAC_{13} and MAC_{31} without possessing correct keys. Initially, the attacker node knows the IDs of its neighbors $\{l_1, l_2, l_3, l_4\}$. The attacker can create new data n, and can append elements of $\{l_1, l_2, l_3, l_4\}$, and n to the end of the ID list it receives. Finally, it can broadcast and unicast its messages to honest nodes.

The attacker overhears only messages sent by its neighbors. The attacker combines this accumulated knowledge and its initial knowledge to perform attacks. Let T_{l^p} be a tuple that consists of the elements in $\{l_1, l_2, l_3, l_4\}$.

Formally, the operation of the attacker node is defined as follows: $P_A \stackrel{def}{=} (\tilde{x})\,.\nu n.\langle f\,(\tilde{x}, T_{l^p}, n)\rangle$, where $\tilde{x}$ is a tuple $(x_1,\ldots,x_m)$ of variables, νn means that the attacker creates new data n. The function $f\,(\tilde{x}, T_{l^p}, n)$ represents the message the attacker generates from (i) the eavesdropped messages that it receives by binding them to $\tilde{x}$, (ii) its initial knowledge and (iii) the newly generated data n, respectively.

[1] Due to page limitation we omit the presence of sequence number and message ID in messages, but we note that our attack works in the same way in the presence of them.

As the next step, we define an *ideal model* of *netw*, denoted as $netw_{spec}$. The definition of $netw_{spec}$ is the same as *netw* except that the desription of N_1 is $\left\lfloor P_1^{spec} \right\rfloor_{l_1}^{\{l_2,l_a\}}$. Process P_1^{spec} models the ideal operation of the source node N_1 in the sense that although the source node does not know the route to the destination it is equipped with the special function *consistent*(*List*) that informs it about the correctness of the returned route.

We define this ideal source node as follow:

$$
\begin{aligned}
P_1^{spec} &\stackrel{def}{=} let\ MAC_{13} = mac\left((l_1, l_3), k(l_1, l_3)\right)\ in\ Init_{spec}.\\
Init_{spec} &\stackrel{def}{=} \langle (req, l_1, l_3, MAC_{13}, list())\rangle.!Rep_{spec}.\\
Rep_{spec} &\stackrel{def}{=} \left(x_{rep}\right).[1(x_{rep}) = l_1]\ [2(x_{rep}) = rep]\\
&\quad [3(x_{rep}) = l_1][4(x_{rep}) = l_3][fst(5(x_{rep})) \in \{l_2, l_a\}]\\
&\quad [mac\left(\left(l_1, l_3, 5(x_{rep})\right), k(l_1, l_3)\right) = 6(x_{rep})]\\
&\quad [consistent\left(5(x_{rep})\right) = true].\langle RouteOK\rangle.0
\end{aligned}
$$

Intuitively, in the ideal model, every route reply that contains a non-existent route is caught and filtered out by the initiator of the route discovery. Therefore, security is achieved by definition.

Next we give the *definition of secure routing* based on labelled bisimilarity:

Definition 2 *A routing protocol is said to be secure if for all extended networks E and its corresponding ideal network E_{spec} where both include the same but arbitrary attacker node, we have: $E \approx_l^N E_{spec}$.*

Theorem 1 *The SRP protocol is insecure.*

Proof 1 ***(Sketch)*** *We will show that $netw \approx_l^N netw_{spec}$ does **not** hold in the presence of the attacker $\mathcal{M}_A$ because the third condition of the Definition 1 is violated. When the attacker node receives the request message (req, l_1, l_3, MAC_{13}, $[l_2]$) from node l_2, it creates some new fake node indentifier n, then it adds n and the identifier l_4 to the list $[l_2]$, then it re-broadcasts (req, l_1,l_3,MAC_{13},$[l_2, n, l_4]$). When this message reaches the target node l_3 it passes all the verifications made by l_3. Then, node l_3 generates the reply (l_4, rep, l_1, l_3,MAC_{31}) and sends it back to l_4. The attacker node overhears this message and forwards it to the source l_1 in the name of l_2. As the result, in netw node l_1 accepts the returned invalid route $[l_2, n, l_4]$ and outputs RouteOK by the $\stackrel{\nu x.\langle x\rangle:\overline{l_1}\{l_2 l_a\}}{\longrightarrow}$ transition step. However, in $netw_{spec}$ node l_1 does not accept the returned route, thus, RouteOK is not output. Formally, at this point $netw_{spec}$ cannot perform the transition $\stackrel{\nu x.\langle x\rangle:\overline{l_1}\{l_2 l_a\}}{\longrightarrow}$, which violates the third condition of Definition 1.*

6 On automating the verification

In this section, we present a novel automated verification technique based on deductive model checking. The novelty of our method compared with the other related works is that it supports explicit modelling of cryptographic primitives which is not the case in [2, 7]. Compared to [3, 11] our method can be fully automated. Our technique was inspired by the concept of the ProVerif automatic verification tool [9], however, as opposed to [9] it is designed for verifying routing protocols and includes numerous novelties such as broadcast communications, neighborhood, and considers an attacker model specific to wireless ad hoc networks.

The reason why we propose a custome verification algorithm and not apply directly existing frameworks, such as the Process Analysis Toolkit (PAT) [20], is that in their current version they are very cumbersome to use for finding these kind of subtle attacks against secure routing protocols. Currently PAT does not support any special module for secure routing protocols. Although PAT supports the CSP module that is based on algebra, the verification engine of CSP is not optimized for verifying secure routing protocols due to the lack of formal syntax and semantics. Furthermore, we found that the verification engine of the ProVerif tool is very effective and closely related to our problem. Finally, this engine can be extended/modified for secure routing protocols in a straightforward way.

6.1 From the process calculus to Horn-clauses

In the ProVerif verification tool [9] the input of the tool is the protocol description in the syntax of the applied π-calculus. The tool then translates this to Prolog rules, which are Horn clauses, to make automatic reasoning. Following this concept, in our verifcation method, routing protocols are specified in the syntax of our calculus. This is then translated to Horn-clauses using translation rules. The reader can find details in [16].

Terms t in the calculus are translated to *patterns* p of Horn-clauses that is defined as follows: Variables x and node IDs l in the calculus are translated to x^p and l^p of Horn clauses, respectively. The attacker node has the unique identifier l_{att}^p. Node identifiers are atomic values (i.e., constants). Data n and function f are encoded as functions with arity k and t: $n[p_1, \ldots, p_k]$, and $f(p_1, \ldots, p_t)$, respectively. Finally, s is a special variable that takes values from the set *{honest, advr}*, where *honest* and *advr* are constants.

The protocol and the computation ability of the attacker are modelled by the implication rules (clauses) of the form $F_1 \wedge \ldots \wedge F_n \rightarrow C$, where F_i $i \in \{1, \ldots, n\}$ and C are facts that are defined as predicate application $pred(p_1,\ldots,p_m)$. The left side of the rule is called as hypothesis while the right side is called as conclusion.

In our method the rules use the *facts* $wm(p)$ and $att(p)$ which model that the wireless medium and the attacker know the pattern p, respectively. The fact *routeok(s)* is used for signalling that the returned reply has been accepted by the source node. We will discuss *routeok(s)* in detail later. In addition, the fact $route(l_{src}^p, l_{dest}^p)$ is used to assign the source node l_{src}^p and the target node l_{dest}^p for a given route discovery. Finally, the fact $nbr(l_i^p, l_j^p)$ models that node l_j^p is in the transmission range of node l_i^p.

6.2 Generating protocol clauses

We exploit the fact that each node behaves in the same way in a route discovery phase of routing protocols. Hence, the operation of every node can be uniformly defined as follows:

$$P_i \stackrel{def}{=} !^{i_1} Init^{P_i} | !^{i_2} Interm^{P_i} | !^{i_3} Dest^{P_i}$$

Every node can start a route discovery towards any other node, in this case the *Init* process is invoked. Every node can be an intermediate node, in which case its *Interm* process is invoked. Finally, every node can be a target node when the *Dest* process is invoked. We note that the specified structure of each process depends on the specified routing protocol. However, in general form P_i can be modelled as:

$$Init^{P_i} \stackrel{def}{=} c_{P_i}(x_{dest}).Prot_{init}.\langle x_{msgreq}\rangle.!^{i_4} Rep^{P_i}_{init}.$$
$$Rep^{P_i}_{init} \stackrel{def}{=} (x_{rep}).Verif_{init}.\langle RouteOK\rangle.$$
$$Interm^{P_i} \stackrel{def}{=} (y_{req}).Prot_{interm}.\langle y_{msgreq}\rangle.!^{i_5} Rep^{P_i}_{interm}.$$
$$Rep^{P_i}_{interm} \stackrel{def}{=} (y_{rep}).Verif_{interm}.\langle y_{msgrep}\rangle.$$
$$Dest^{P_i} \stackrel{def}{=} (z_{req}).Verif_{dest}.\langle z_{msgrep}\rangle.$$

Process $Init^{P_i}$ receives on the channel c_{P_i} the ID of the destination node to which it starts the route discovery. Then P_i goes through a protocol dependent processing part $Prot_{init}$, followed by broadcasting the request x_{msgreq} and then it waits for the reply at the end.

On receiving a reply x_{rep}, process $Rep^{P_i}_{init}$ does verification steps in a protocol dependent manner and then signals the term *RouteOK* if x_{rep} passes all verifications.

Process $Interm^{P_i}$ describes the case when P_i is an intermediate node and says that when P_i receives some request y_{req} it goes through a protocol dependent processing part $Prot_{interm}$, and re-broadcasts the request y_{msgreq}. Finally, it waits for the reply.

On receiving a reply y_{rep}, process $Rep^{P_i}_{interm}$ does verification steps in a protocol dependent manner and then forwards the (may be modified) reply y_{msgrep}.

Process $Dest^{P_i}$ describes the case when P_i is a destination node and says that when P_i receives some request z_{req} it does verification steps and then sends back the reply z_{msgrep}.

We assume that the reply messages x_{rep}, y_{rep}, y_{msgrep}, and z_{msgrep} include the information of the addressee. Hence, when a node receives a reply it can check whether the reply is intended to it.

The exclamation mark $!^i$ models replication, where i is a session identifier to distinguish different process instances in replication. Session identifier further is used to track attack traces.

Process P_i is then translated to the protocol rules. A routing protocol modelled by P_i is specified in the following clauses that serve as a template of the correct operation of the routing protocol:

(Protocol rules: Template of the correct operation) ::=

$R^{req}_1.\ route(x^p_{src}, x^p_{dest}) \xrightarrow{p,i_1} wm(x^p_{msgreq})$

$R^{rep}_1.\ wm((x^p_{rep}, s)) \wedge Verif^{facts}_{init} \xrightarrow{p,i_2}$ **routeok(s)**.

$R^{req}_2.\ wm(y^p_{req}) \wedge nbr(y^p_{from}, y^p_{to}) \xrightarrow{p,i_3} wm(y^p_{msgreq})$.

$R^{rep}_2.\ wm((y^p_{rep}, s)) \wedge Verif^{facts}_{interm} \xrightarrow{p,i_4} wm((y^p_{msgrep}, s))$.

$R^{req}_3.\ wm(z^p_{req}) \wedge nbr(z^p_{from}, l^p_{dest}) \wedge Verif^{facts}_{dest} \xrightarrow{\tilde{p},i_5}$
$wm((z^p_{msgrep}, honest))$

When a message is received by an attacker node:

$R^{att}_1.\ wm(y^p_{req}) \wedge nbr(y^p_{from}, l^p_{att}) \xrightarrow{p,i_6} att(y^p_{req})$

$R^{att}_2.\ wm(y^p_{rep}) \wedge nbr(y^p_{from}, l^p_{att}) \xrightarrow{p,i_7} att(y^p_{rep})$

We note that these rules represent the protocol independent form, however, in specified protocols such as SRP the translation will take into account the *Verif* and *ProtDep* parts. This may yield additional rules, and hypotheses. See Section 6.6 for examples. $Verif^{facts}_{init}$, $Verif^{facts}_{interm}$ and $Verif^{facts}_{dest}$ are set up by a conjunction of fact(s).

We assume that the route request and reply messages x^p_{rep}, y^p_{req}, y^p_{rep} and z^p_{req} include identifiers of the nodes from which they are received. The identifiers are specified by the patterns x^p_{from}, y^p_{from}, and z^p_{from}.

Rules R^{req}_1 and R^{rep}_1 model the operation of the source node. In particular, rule R^{req}_1 says that the source node x^p_{src} generates the request message x^p_{msgreq} and then broadcasts it. Rule R^{rep}_1 says that if the source node receives a reply message and if all verification steps the source node made on the reply pass then the returned route is accepted, this is modelled by the derivation of the fact **routeok(s)**. The variable s is a flag that takes its value from the set *{honest, advr}*, and is used to determine whether the attacker node intercepts a reply message (when $s = advr$) during the route discovery or not (when $s = honest$). More precisely, the derivation of the fact **routeok(honest)** means that the request or reply are modified by honest nodes only, while when **routeok(advr)** is derived the messages are manipulated by the attacker node. The variable s is appended after each reply message forming the tuples (x^p_{rep}, s), (y^p_{rep}, s), (y^p_{msgrep}, s), and $(z^p_{msgrep}, honest)$.

Rules R^{req}_2 and R^{rep}_2 model the operation of intermediate nodes. In particular, rule R^{req}_2 says that if the node y^p_{to} is in the transmission range of the y^p_{from} then node y^p_{to} receives the request y^p_{req} sent by y^p_{from}. The messages y^p_{req} and y^p_{msgreq} could be the same depending on the specified protocol. After processing the request y^p_{req} node y^p_{to} re-broadcasts it. Rule R^{rep}_2 says that if an intermediate node receives the message tuple (y^p_{rep}, s) then it makes verification steps. If all verifications pass it appends s unchanged to the reply y^p_{msgrep} then forwards the message tuple (y^p_{msgrep}, s). Again, the messages y^p_{rep} and y^p_{msgrep} could be the same depending on the specified protocol.

The rule R^{req}_3 models the operation of the destination node and says that if the destination node l^p_{dest} is the neighbor of some honest intermediate node z^p_{from} then the destination receives the request z^p_{req}. If all the verifications made by the destination pass it generates a reply z^p_{msgrep} and adds the flag *honest* signalling

that the attacker still has not seen the reply. Finally, it sends back the tuple $(z^p_{msgrep}, honest)$.

Rules R^{att}_1 and R^{att}_2 concern the case when the attacker node l^p_{att} intercepts the request y^p_{req} and reply y^p_{rep} because it is a neighbor of some honest node y^p_{from}.

6.3 Knowledge and computation ability of the attacker

The ability of an attacker node is represented in the following rules. These rules represent the strongest actions that can be performed by the attacker node l^p_{att}.

$(Init.\ knowl.) ::= \forall l^p_n$ neighbors of l^p_{att}:
$I_1.\ att(l^p_{att}), att(l^p_n);\ I_2.\ att(k(l_{att}, l^p_n));\ I_3.\ nbr(l_{att}, l^p_n).$

I_1 means that initially the attacker knows its own ID and the ID of its honest neighbors, I_2 means that the attacker possesses all the keys it shares with the honest nodes. Finally, I_3 means that the attacker is aware of its neighborhood.

In addition, we define a strong computation ability for the attacker node as follows:

(Computation ability - protocol independent) ::=

$C_1.att(i) \rightarrow att(n[i])$

C_2.*For each public function* f *of n-arity*

$att(x^p_1) \wedge \ldots att(x^p_n) \rightarrow att(f(x^p_1, \ldots, x^p_n))$

C_3.*For each public function* g *that*

$g(f(x^p_1, \ldots, x^p_n), y^p) \rightarrow x^p$:

$att(f(x^p_1, \ldots, x^p_n)) \wedge att(y^p) \rightarrow att(x^p)$

$C^1_4.att(y^p_{req}) \wedge nbr(l^p_{att}, y^p_{to}) \wedge Verif^{facts}_{att}$

$\xrightarrow{p,i_8} wm(y^p_{msgreq})$.

$C^2_4.att((y^p_{rep}, s)) \wedge nbr(l^p_{att}, y^p_{to}) \wedge Verif^{facts}_{att}$

$\xrightarrow{p,i_9} wm((y^p_{msgrep}, advr))$.

$C^3_4.att(z^p_{req}) \wedge nbr(l^p_{att}, l^p_{dest}) \wedge nbr(l^p_{dest}, l^p_{att}) \wedge Verif^{facts}_{att}$

$\xrightarrow{p,i_{10}} att((z^p_{msgrep}, honest))$.

$C^4_4.att((x^p_{rep}, s)) \wedge nbr(l^p_{att}, l^p_{src}) \wedge Verif^{facts}_{att}$

$\xrightarrow{p,i_{11}}$ **routeok(advr)**.

Rule C_1 means that the attacker node can create arbitrary new data n such as fake identifiers, where i is a session ID to identify that the data is created in which protocol run. Rule C_2 means that the attacker can generate arbitrary messages based on its actual knowledge. Rule C_3 means that the attacker can perform computation on function f. For instance, if f is a digital signature *sign* then its corresponding "inverse" function g, which is *checksign* can be performed to verify the signature. The set of functions depends on the protocol we are examining.

The rules C^1_4, C^2_4, C^3_4 and C^4_4 say that the attacker can broadcast the messages it has. Rule C^1_4 describes the case in which the attacker broadcasts the request y^p_{req} that is received by its neighbor y^p_{to} who then outputs some message y^p_{msgreq} if all verifications it made on y^p_{req} (modelled by $Verif^{facts}_{att}$) pass. Rule C^2_4 says that the reply y^p_{req} forwarded by the attacker is overheared by its neighbor y^p_{to} who then forwards the reply with the flag *advr* signalling that the reply is forwarded by the attacker. Rule C^3_4 concerns the scenario when the destination node l^p_{dest} and the attacker l^p_{att} are neighbors of each other and says that if the request z^p_{req} broadcasted by l^p_{att} passes all the required verifications then the destination sends back a reply message along with the flag *honest* signalling that the reply is just generated and has not been seen by the attacker. Finally, rule C^4_4 concerns the scenario when the source node l^p_{src} is a neighbor of the attacker l^p_{att} and says that if the reply z^p_{rep} forwarded by l^p_{att} passes all the required verifications made by l^p_{src} then the reply is accepted. The implication $\xrightarrow{p,i}$ includes the message and session ID that required for tracking attacks.

6.4 Deductive algorithm

The operation of a routing protocol is modelled by resolution steps, defined as follows:

Definition 3 *Given two rules* $r_1 = H_1 \rightarrow C_1$, *and* $r_2 = F \wedge H_2 \rightarrow C_2$, *where* F *is* ***any*** *hypothesis of* R_2, *and* F *is unifiable with* C_1 *with the most general unifier* σ, *then the resolution* $r_1 \circ_F r_2$ *of them yields a new rule* $H_1\sigma \wedge H_2\sigma \rightarrow C_2\sigma$.

Let us recall the example in Fig. 1. The resolution $route(l^p_1, l^p_3) \circ_F R_1$, where p=(*req, x_{src}, x_{dest}, ID, MAC*) and $F = route(x^p_{src}, x^p_{dest})$ yields the fact wm(*req,* l^p_1, l^p_3, *ID, MAC*) with the unifier $\{x^p_{src} \leftarrow l^p_1,\ x^p_{dest} \leftarrow l^p_3\}$. This resolution step models the broadcasting of the route request message (*req, x_{src}, x_{dest}, ID, MAC*) generated by node l^p_1.

In the following we give a sketch of the deductive algorithm of our method. More details can be found in [16].

Algorithm 1 *The input of the algorithm is the tuple* $(\mathcal{T}_0, \{\mathcal{N}_{l^p_n}\}, \{\mathcal{N}_{l^p_{att}}\}, route(l^p_{src}, l^p_{dest}), \mathcal{K}, \mathcal{A}) : \mathcal{T}_0$ *is the set of protocol rules; the two sets* $\{\mathcal{N}_{l^p_n}\}$ *and* $\{\mathcal{N}_{l^p_{att}}\}$ *specify the neighborhood of the honest node* l^p_n *and the attacker node* l^p_{att} *which are given by the set of facts* $nbr(l^p_n, l^p_i)$ *and* $nbr(l^p_{att}, l^p_j)$, *respectively; the fact* $route(l^p_{src}, l^p_{dest})$ *specifies the source and target nodes for a given route discovery;* $\mathcal{A}$ *and* $\mathcal{K}$ *are the sets of attacker computation rules and the knowledge base of the attacker, respectively.*

The algorithm is based on the breadth-first search to reach the destination node l^p_{dest} *from the source* l^p_{src}. *Each node broadcasts its message to neighbors. This is modelled by resolution steps at each node. The algorithm starts with the resolution* $route(l^p_{src}, l^p_{dest}) \circ_{route(x^p_{src}, x^p_{dest})} R^{req}_1$, *which says that the source node broadcasts the request. The request is then forwarded by intermediate nodes, which are modelled by the resolutions of rules* R^{req}_2 *or* R^{req}_3 *in* $\mathcal{T}_0$ *with the conclusions resulted in the previous resolutions.*

When the destination node receives a request it sends back a reply if the verifications it made on the request were successful, this scenario is modelled by the rule R^{req}_3.

The backward propagation of the reply is modelled by the resolution steps between the results of a previous resolution with the rules R_2^{rep}, R_1^{rep}. Resolutions with rule R_2^{rep} means that the reply is returned to intermediate nodes, while resolutions with rule R_1^{rep} model the scenario in which the reply reaches the source node and is accepted.

When the attacker node intercepts a message p broadcasted by one of its neighbors (happens when resolutions made by the algorithm includes R_1^{att} or R_2^{att}) it adds the new fact att(p) into $\mathcal{K}$. This step represents that the attacker collects information during the route discovery. After updating its knowledge, the attacker attempts to construct incorrect messages and forwards them to honest nodes based on the collected information $\mathcal{K}$ and its computation rules $\mathcal{A}$.

Theorem 2 *If there is an attack for a given network topology $\mathcal{N}$ and tuple (l_s^p, l_d^p, l_{att}^p) then the algorithm will find some attack.*

The proof of this theorem is complicated and we need to take into account auxiliary definitions and theorems. Due to page limitation we refer the reader to [16] for details. In the next theorem we discuss the complexitiy of the algorithm. Because the algorithm is mostly based on resolution steps, below we provide the complexity by upper bound the number of total resolution steps.

Theorem 3 *Let us assume that $Verif_{init}^{facts}$, $Verif_{interm}^{facts}$ and $Verif_{dest}^{facts}$ are empty. The number of resolution steps made by the algorithm is upper bounded by $(\mathcal{D}+2)(|N|+|E|)+\mathcal{B}$, where $|N|$, $|E|$ are the number of nodes and edges of the topology; $\mathcal{D}$ is the total number of incorrect messages created by the attacker during the algorithm that are accepted by its honest neighbors; and $\mathcal{B}$ is the total number of the resolution steps (computation steps) made by the attacker during the algorithm. For practical reasons both $\mathcal{D}$ and $\mathcal{B}$ are assumed to be finite.*

Proof 2 ***(sketch)*** *The propagation of a request from the source to the destination can be seen as a breadth-first traversing in the graph. In case there is no attacker in the network, the algorithm performs at most $|N|+|E|$ resolution steps. The same is true regarding the propagation of the reply. Note that in most cases the reply is returned by unicast sending instead of broadcast, thus, the number of resolution steps equals to the length of the route. Taking into account the attacker node, whenever it intercepts a new request/reply it attempts to compute incorrect messages that will be accepted by its honest neighbors. These incorrect messages will propagate to the source/destination, which takes at most $\mathcal{D}(|N|+|E|)$ resolution steps. Finally, the total resolution steps made by the attacker to construct incorrect messages is $\mathcal{B}$.*

We note that in case $Verif_{init}^{facts}$, $Verif_{interm}^{facts}$ and $Verif_{dest}^{facts}$ are not empty, the complexity depends on the maximal number and the type of the facts they contain. In a general case the complexity can be exponential, however, by taking into account the property of the specified protocols and properly restricting the attacker ability this complexity can be reasonably reduced as in case the SRP protocol.

6.5 Derivability and derivation tree

In order to give the definition of derivability and graphical representation of the reasoning in the previous subsection we introduce the notion of *derivation tree*.

Definition 4 *We say that F is a closed fact if it does not contain any variable.*

For example *routeok(honest)*, *routeok(advr)* and $route(l_{src}^p, l_{dest}^p)$ are closed facts. However, $route(x_{src}^p, x_{dest}^p)$ is not closed.

Definition 5 *Let F be a closed fact. Let C be a set of clauses. A derivation tree of F from C is a finite tree defined as follows:*

1 *Its nodes are labelled by the name of the rule that is applied in the resolution. In our case these rules can be the protocol rules $R_1^{req}, \ldots, R_3^{req}$, R_1^{att}, R_2^{att}, or the attacker rules I_1, I_2, I_3, C_1, C_2, C_3, C_4^1, C_4^2, C_4^3, and C_4^4. In addition, nodes can be captioned by the pair (p, i).*

2 *Its edges are labelled by the facts using in the resolution steps. Incoming edges represent the hypotheses of the rule that is applied in the resolution steps while the outgoing edge represents the conclusion of the rule.*
If the tree contains a node such that:
(i) it is labelled by the rule R, where $R = H_1' \wedge \ldots \wedge H_n' \xrightarrow{p,i} C'$, H_i' $(1 \le i \le n)$ and C' are not closed facts; (ii) it has the incoming edges labelled by $H_1, H_2, \ldots, H_n$ and an outgoing edge C, where H_i $(1 \le i \le n)$ and C are closed facts.
Then the n resolutions $H_i \circ_{H_i'} R$ of the n facts H_i $(1 \le i \le n)$ and the rule R are successful with the unifiers σ_i $(1 \le i \le n)$, and the result of the resolutions is the conclusion C, where $C = C'\sigma_1 \ldots \sigma_n$.
In short, this means that the closed fact C can be derived from $H_1, H_2, \ldots, H_n$ using rule R.

The right side of the Fig. 2 shows the resolution $route(l_s^p, l_d^p) \circ_{route(x_s^p, x_d^p)} R$, where R is the rule $route(x_s^p, x_d^p) \xrightarrow{p,i} wm(x_s^p, x_{msgreq}^p)$.

6.6 Application of our automatic verification method

Next, we give an overview of how our verification method works in case of the SRP protocol considering the scenario in Section 2.

The node identifiers of the five nodes are the constants l_1^p, l_2^p, l_3^p, l_4^p, and l_{att}^p. The input of the algorithm is the tuple ($\mathcal{T}_0$, $\{\mathcal{N}_{l_i^p} | 1 \le i \le 4\}$, $\mathcal{N}_{l_{att}^p}$, $route(l_1^p, l_3^p)$, $\mathcal{K}$, $\mathcal{A}$) where: $\mathcal{T}_0$ contains 11 rules in total including the rules R_1^{req}, R_2^{req} and R_1^{att} that we use in this demonstration.

The topology is defined by the five sets

$$\mathcal{N}_{l_1^p}, \mathcal{N}_{l_2^p}, \mathcal{N}_{l_3^p}, \mathcal{N}_{l_4^p}, \text{ and } \mathcal{N}_{l_{att}^p}$$

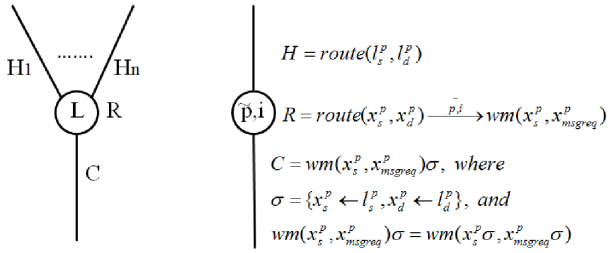

Fig. 2. On the left: The derivation of the closed fact C from $H_1, H_2, \ldots, H_n$ by using rule R, where $R = H_1' \wedge \ldots \wedge H_n' \xrightarrow{p,i} C'$, H_i' $(1 \le i \le n)$ and C' are not closed facts
On the right: The derivation of the fact $wm(x_s^p, x_{msgreq}^p)\sigma$, where $\sigma = \{x_s^p \leftarrow l_s^p, x_d^p \leftarrow l_d^p\}$, from $route(l_s^p, l_d^p)$ using rule $route(x_s^p, x_d^p) \xrightarrow{p,i} wm(x_s^p, x_{msgreq}^p)$.

where :

$$\begin{aligned}
\mathcal{N}_{l_1^p} &= \left\{nbr(l_1^p, l_2^p), nbr(l_1^p, l_{att}^p)\right\};\\
\mathcal{N}_{l_2^p} &= \{nbr(l_2^p, l_1^p), nbr(l_2^p, l_{att}^p)\};\\
\mathcal{N}_{l_3^p} &= \{nbr(l_3^p, l_{att}^p), nbr(l_3^p, l_4^p)\};\\
\mathcal{N}_{l_4^p} &= \left\{nbr(l_4^p, l_{att}^p), nbr(l_4^p, l_3^p)\right\};\\
\mathcal{N}_{l_{att}^p} &= \left\{nbr(l_{att}^p, l_1^p), nbr(l_{att}^p, l_2^p),\right.\\
&\quad \left. nbr(l_{att}^p, l_3^p), nbr(l_{att}^p, l_4^p)\right\}
\end{aligned}$$

The initial knowledge $\mathcal{K}$ and the computational ability $\mathcal{A}$ of the attacker are defined by the two sets $\{I_1, I_2, I_3\}$ and $\{S_1, S_2, S_3, S_4, S_5\}$, respectively. The initial knowledge is the same as in Section 6.3. The rules S_1, S_2, S_3, S_4 and S_5 are specified as follows:

$$S_1.att(i) \rightarrow att(n[i])$$

Rule S_1 is the same as rule C_1 in Section 6.3.

$$\begin{aligned}
S_2.\ & att\left((req^p, x_{src}^p, x_{dest}^p, x_{ID}^p, x_{mac}^p, List^p)\right) \wedge att(y_l^p)\\
& \rightarrow att((req^p, x_{src}^p, x_{dest}^p, x_{ID}^p, x_{mac}^p, [List^p, y_l^p]))
\end{aligned}$$

Rule S_2 is the special case of rule C_2 and says that the attacker node can append any data it has to the end of the ID list embedded in the request it receives. Pattern $List^p$ represents an ID list, which can be empty. The variables x_{src}^p, x_{dest}^p, x_{ID}^p, and x_{mac}^p specify the ID of the source and destination node, the message ID, and the message authentication code, respectively.

$$\begin{aligned}
S_3.\ & att((x_l^p, rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, List^p, x_{mac}^p)) \wedge att(y_l^p)\\
& \rightarrow att((y_l^p, rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, List^p, x_{mac}^p))
\end{aligned}$$

Rule S_3 is an another special case of rule C_2 and says that if the attacker receives a reply message $(rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, List^p, x_{mac}^p)$ addressed to node x_l^p then it can replace the node identifier at the beginning of the message, which specifies the addressee, by an another identifier y_l^p. This rule intend to model that the attacker can forward the reply in the name of another nodes.

$$\begin{aligned}
S_4.\ & att\left(req^p, x_{src}^p, x_{dest}^p, x_{ID}^p, MAC^{req}, [List^p, x_{prev}^p]\right)\\
& \wedge nbr(l_{att}^p, x_{dest}^p) \wedge nbr(x_{dest}^p, l_{att}^p)\\
& \wedge nbr(x_{dest}^p, x_{prev}^p) \xrightarrow{p,i_{10}}\\
& att\left((x_{prev}^p, rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, [List^p, x_{prev}^p],\right.\\
& \left. MAC_1^{rep}), honest\right).
\end{aligned}$$

where MAC^{req} is

$$mac((rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, k(x_{src}^p, x_{dest}^p))$$

and MAC_1^{rep} is

$$mac\left((rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, [List^p, x_{prev}^p]), k(x_{src}^p, x_{dest}^p)\right).$$

Rule S_4 is the special case of rule C_4^3 concerning the case when the destination node and the attacker node are neighbors of each other. In this special case $Verif_{att}^{facts}$ is $nbr(x_{dest}^p, x_{prev}^p)$ that models the verification step in which the destination checks if the last ID in the ID list belongs to its neighbor.

$$\begin{aligned}
S_5.& att\left((_{src}^p, rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, [x_{next}^p, List^p],\right.\\
& \left.\mathrm{MAC}_2^{rep}), s\right) \wedge nbr(l_{att}^p, x_{src}^p) \wedge nbr(x_{src}^p, x_{next}^p)\\
& \xrightarrow{p,i_{11}} \textbf{routeok(advr)}.
\end{aligned}$$

where MAC_2^{rep} is

$$mac\left((rep^p, x_{src}^p, x_{dest}^p, x_{ID}^p, List^p), k(x_{src}^p, x_{dest}^p)\right).$$

Rule S_5 is the special case of rule C_4^4 concerning the case when the source node is a neighbor of the attacker node.

The most important protocol rules that we use in this demonstration are the rules R_1^{req}, R_2^{req} and R_1^{att}:

$$\begin{aligned}
R_1^{req} = & route(x_{src}^p, x_{dest}^p) \xrightarrow{\tilde{p},i_1}\\
& wm(req^p, x_{src}^p, x_{dest}^p, ID,\\
& mac((req^p, x_{src}^p, x_{dest}^p, ID, k(x_{src}^p, x_{dest}^p)), [\,])
\end{aligned}$$

Rule R_1^{req} models the scenario when the source node x_{src}^p creates and broadcasts the request message

$$\begin{aligned}
& \left(req^p, x_{src}^p, x_{dest}^p, x_{ID}^p,\right.\\
& \qquad \left. mac\left((req^p, x_{src}^p, x_{dest}^p, x_{ID}^p), k(x_{src}^p, x_{dest}^p)\right)\right).
\end{aligned}$$

$$\begin{aligned}
R_2^{req} = & wm\left((req^p, y_{src}^p, y_{dest}^p, y_{ID}^p, y_{mac}^p, [\,])\right) \wedge\\
& nbr(y_{src}^p, y_{l_x}^p) \xrightarrow{p,i_2}\\
& wm\left((req^p, y_{src}^p, y_{dest}^p, y_{ID}^p, y_{mac}^p, [y_{l_x}^p])\right)
\end{aligned}$$

Rule R_2^{req} says that if the route request message is broadcasted by the source node then the honest neighbors of the source node

receive the request and they append their own identifier to the request and re-broadcast the modified request.

$$R_1^{att} = wm\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}])\right) \wedge nbr(y^p_{prev}, l^p_{att}) \xrightarrow{p,i_6} att\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}])\right)$$

Rule R_1^{att} says that if a route request message is broadcasted by an intermediate node and the attacker node is its neighbor then the attacker intercepts that request. $List^p$ is a pattern that represent a list of node identifiers, which can be empty.

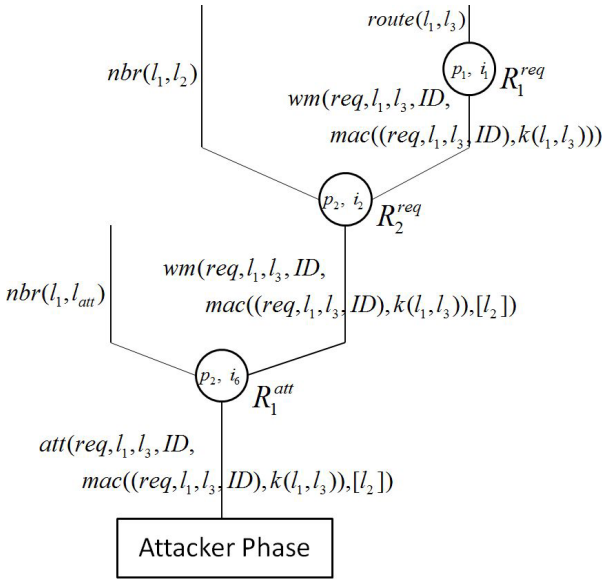

Fig. 3. This part of the derivation tree describes the resolution steps that represent the propagation of the request message from the source node l_1^p to the attacker node l_{att}^p.

The derivation of one possible attack against the SRP protocol is modelled by derivation trees shown in the Fig. 3 and Fig. 4. Fig. 3 describes the propagation of the request message from the source node l_1^p to the attacker node l_{att}^p via the intermediate node l_2^p. First, the resolution $route(l_1^p, l_3^p) \circ_{route(x^p_{src}, x^p_{dest})} R_1^{req}$ is computed. With the unifier σ_1,

$$\sigma_1 = \{x^p_{src} \leftarrow l_1^p, x^p_{dest} \leftarrow l_3^p\} route(l_1^p, l_3^p) \circ_{route(x^p_{src}, x^p_{dest})} R_1^{req}$$

yields the fact

$$wm((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [\,]),$$

which is then resolved with R_2^{req} and yields the rule R_{tmp}^1:

$$R^1_{tmp} = nbr(l_1^p, y^p_{l_x}) \xrightarrow{p,i_2} wm\left((req^p, l_1^p, l_3^p, ID^p, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [y^p_{l_x}])\right)$$

where the unifier σ_2 of the above resolution is

$$\sigma_2 = \{y^p_{src} \leftarrow l_1^p, y^p_{dest} \leftarrow l_3^p, y^p_{ID} \leftarrow ID, y^p_{mac} \leftarrow mac\left((req^p, l_1^p, l_3^p, ID), k(l_1^p, l_3^p)\right)\}.$$

Afterwards, the facts in the set $\mathcal{N}_{l_1^p}$ are resolved with R_{tmp}^1. The resolution $nbr(l_1^p, l_2^p) \circ_{nbr(l_1^p, y^p_{l_x})} R^1_{tmp}$ with the unifier σ_3, $\sigma_3 = \{y^p_{l_x} \leftarrow l_2^p\}$ yields the fact $wm((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p]))$. Intuitively, this means that node l_2^p received the request message broadcasted by l_1^p, and then l_2^p appends its identifier to the request and re-broadcasts it.

The following resolution steps model the case when the attacker node intercepts the request broadcasted by node l_2^p:

The resolution

$$wm((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])) \circ_F R_2^{att}$$

where F is

$$wm\left((req^p, y^p_{src}, y^p_{dest}, y^p_{ID}, y^p_{mac}, [List^p, y^p_{prev}])\right)$$

and the the unifier

$$\sigma_4 \text{ is } \sigma_2 \cup \{List^p \leftarrow [\,], y^p_{prev} \leftarrow l_2^p\}.$$

As the result we get the rule R_{tmp}^2:

$$R^2_{tmp} = nbr(l_2^p, l_{att}^p) \xrightarrow{p,i_6} att\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])\right)$$

Finally, the algorithm searches for the fact $nbr(l_2^p, l_{att}^p)$ in the set $\mathcal{N}_{l_2^p}$. When $nbr(l_2^p, l_{att}^p)$ is found the resolution $nbr(l_2^p, l_{att}^p) \circ_{nbr(l_2^p, l_{att}^p)} R^2_{tmp}$ is computed, which yields the fact

$$att\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])\right).$$

Fig. 4 describes the behaviour of the attacker node after intercepting the message $(req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])$: First, the attacker creates a fake identifier $n[i]$ by rule S_1. Afterwards, the fake identifier is appended to the ID list $[l_2^p]$, this step is modelled by the two resolutions

$$att\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])\right) \circ_F S_2 \text{ and } att(n[i]) \circ_{att(y^p)} R^3_{tmp}$$

where R^3_{tmp} is the result of the first resolution.

Thereafter, the attacker appends the identifier l_4^p to the list $[l_2^p, n[i]]$. This is modelled by the two resolutions

$$att\left((req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p, n[i]])\right) \circ_F S_2 \text{ and } att(l_4^p) \circ_{att(y^p)} R^4_{tmp}$$

where R^4_{tmp} is the result of the first resolution.

We note that $att(l_4^p)$ is an element of the set I_1 that is a part of the initial knowledge of the attacker node.

Then the attacker node broadcasts the modified request, which is received by the destination node l_3^p. The destination node accepts the message sent by the attacker node and generates the reply message. Afterwards, the destination node sends

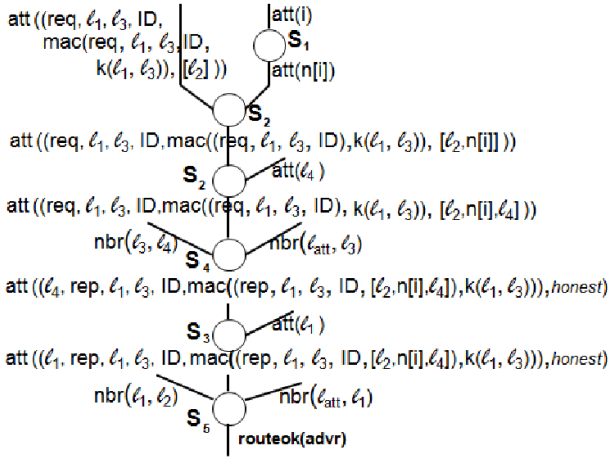

Fig. 4. *The behaviour of the attacker node after intercepting the message* $(req^p, l_1^p, l_3^p, ID, mac((l_1^p, l_3^p, ID), k(l_1^p, l_3^p)), [l_2^p])$.

back the reply to the node l_4^p, which is overheared by the attacker node. This step is modelled by the resolutions involving the rule S_2. On receiving the reply the attacker replaces l_4^p by l_1^p. This step is modelled by the resolution steps involving the rule S_3. Finally, the attacker node forwards the reply to the source node in the name of the node l_2^p. This step is modelled by resolutions using the rule S_5.

7 Conclusion

We argued that designing secure ad-hoc network routing protocols requires a systematic approach which minimizes the number of mistakes made during the design. To this end, we proposed a formal verification method for secured ad-hoc network routing protocols, which is based on a novel process calculus and a deductive proof technique. Our method has a clear syntax and semantics, and it can be fully automated; this latter being a distinctive feature among other formal approaches for verifying secure ad-hoc network routing protocols.

The work described in this paper is work in progress, and we are currently extending it in many ways. The two most important future work items are (i) the development of a fully automated protocol verification software tool based on the theoretical foundations described in this paper, and (ii) the extension of the described verification method to handle arbitrary network topologies and arbitrary number of attacker nodes.

References

1 **Godskesen J Chr**, *A Calculus for Mobile Ad Hoc Networks*, COORDINATION, 2007, pp. 132-150.

2 **Singh A, Ramakrishnan C R, Smolka S A**, *A process calculus for Mobile Ad Hoc Networks*, Sci. Comput. Program. **75** (2010), no. 6, 440–469, DOI 10.1016/j.scico.2009.07.008.

3 **Buttyán L, Vajda I**, *Towards provable security for ad hoc routing protocols*, SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, 2004, pp. 94–105, DOI 10.1145/1029102.1029119.

4 **Milner R, Parrow J, Walker D**, *A calculus of mobile processes, parts I and II*, Information and Computation (September 1992).

5 **Abadi M, Gordon A**, *A calculus for cryptographic protocols: the Spi calculus*, SRC RR 149, January, 1998.

6 **Hu Y.-C., Perrig A**, *A Survey of Secure Wireless Ad Hoc Routing*, IEEE Security and Privacy **2** (2004), no. 3, 28–39.

7 **Todd R A, Yasinsac A**, *Automated Evaluation of Secure Route Discovery in MANET Protocols*, SPIN '08: Proceedings of the 15th international workshop on Model Checking Software (2008), 26–41.

8 **Marshall J D, Xin Yuan**, *An Analysis Of The Secure Routing Protocol For Mobile Ad Hoc Network Route Discovery: Using Intuitive Reasoning And Formal Verification to identify flaws*, THE FLORIDA STATE UNIVERSITY, 2003.

9 **Blanchet B**, *Automatic Proof of Strong Secrecy for Security Protocols*, IEEE Symposium on Security and Privacy, May 2004, pp. 86–100, DOI 10.1109/SECPRI.2004.1301317.

10 **Fournet C, Abadi M**, *Mobile Values, New Names, and Secure Communication*, In Proceedings of the 28th ACM Symposium on Principles of Programming, POPL'01 (2001), 104–115.

11 **Ács G, Buttyán L, Vajda I**, *Provable security of on-demand distance vector routing in wireless ad hoc networks*, In In Proceedings of the Second European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS 2005 **1** (2005), 113–127, DOI 10.1007/11601494_10.

12 **Ács G, Buttyán L, Vajda I**, *Provably Secure On-demand Source Routing in Mobile Ad Hoc Networks*, IEEE Transactions on Mobile Computing **5** (2006), no. 11, DOI 10.1109/TMC.2006.170.

13 **Ács G, Buttyán L, Vajda I**, *The security proof of a link-state routing protocol for wireless sensor networks*, IEEE Workshop on Wireless and Sensor Networks Security **1** (2007), DOI 10.1109/MOBHOC.2007.4428765.

14 **Papadimitratos P, Haas Z J**, *Secure Routing for Mobile Ad Hoc Networks*, Networks and Distributed Systems Modeling and Simulation (2002).

15 **Johnson D, Maltz D**, *Dynamic source routing in ad hoc wireless networks.*, Mobile Computing **1** (1996), DOI 10.1007/978-0-585-29603-6_5.

16 **Buttyán L, Thong T V**, *Formal verification of secure ad-hoc network routing protocols using deductive model-checking*, Technical Report (2010), `http://www.crysys.hu/tvth/adhocTechreport2010.pdf`.

17 **Godskesen J Chr**, *A Calculus for Mobile Ad-hoc Networks with Static Location Binding*, Electron. Notes Theor. Comput. Sci. **242** (2009), no. 1, 161–183, DOI 10.1016/j.entcs.2009.06.018.

18 **Wibling O, Parrow J, Pears A**, *Automatized verification of ad hoc routing protocols*, Formal Techniques for Networked and Distributed Systems FORTE **1** (2004), 343–358, DOI 10.1007/978-3-540-30232-2_22.

19 **Saksena M, Wibling O, Jonsson B**, *Graph grammar modeling and verification of ad hoc routing protocols* **1** (2008), DOI 10.1007/978-3-540-78800-3_3.

20 **Jin Song Dong, Jun Sun, Yang Liu**, *Tutorial: Introduction to PAT Framework*, The 17th International Symposium on Formal Methods (2011).